42장 비동기 프로그래밍

42.1 동기 처리와 비동기 처리

- 함수 실행 과정
  - 함수를 호출하면 함수 코드가 평가되어 함수 실행 컨텍스트가 생성
  - 이때 생성된 함수 실행 컨텍스트는 실행 컨텍스트 스택(콜 스택)에 푸시되고 함수 코드가 실행
  - 함수 코드의 실행이 종료하면 함수 실행 컨텍스트는 실행 컨텍스트 스택에서 팝되어 제거

【 예제 42-01 】

```
const foo = () => {};
const bar = () => {};

foo();
bar();
```

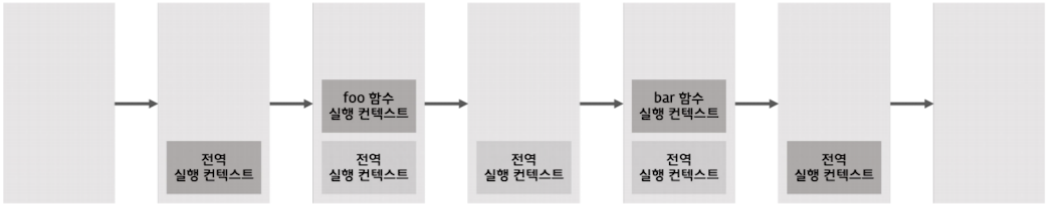

그림 42-1 실행 컨텍스트 스택

- 함수가 호출된 순서대로 순차적으로 실행되는 이유는,

  함수가 호출된 순서대로 함수 실행 컨텍스트가 실행 컨텍스트 스택에 푸시되기 때문

✅ 자바스크립트 엔진은 단 하나의 실행 컨텍스트 스택을 갖는다.

- 즉, 2개 이상의 함수 동시 실행 불가
- 따라서 현재 실행 중인 함수가 종료하면 비로소 다른 함수가 실행되기 시작

✅자바스크립트 엔진은 한 번에 하나의 태스크만 실행할 수 있는 싱글 스레드 방식으로 동작한다.

- 싱글 스레드 방식은 한 번에 하나의 태스크만 실행할 수 있기 때문에

  처리에 시간이 걸리는 태스크를 실행하는 경우 블로킹(작업 중단)이 발생

- ex) 1번 함수가 실행되고나서 3초 후에 2번 함수가 실행되고, 그 후에 3번 함수가 실행되도록 한다면 3번 함수는 그 시간 동안 호출되지 못하고 블로킹된다.

> **동기 처리**
>
> 현재 실행 중인 태스크가 종료할 때까지 다음에 실행될 태스크가 대기하는 방식
> 장점 : 태스크를 순서대로 하나씩 처리하므로 실행 순서가 보장
> 단점 : 앞선 태스크가 종료할 때까지 이후 태스크들이 블로킹

> **비동기 처리**
> 실행 중인 태스크가 종료되지 않은 상태라 해도 다음 태스크를 곧바로 실행하는 방식
> 장점 : 현재 실행 중인 태스크가 종료되지 않은 상태라 해도 다음 태스크를 곧바로 실행하므로
> 블로킹이 발생하지 않음
> 단점 : 태스크의 실행 순서가 보장되지 않음

- 비동기 처리를 수행하는 비동기 함수는 전통적으로 콜백 패턴을 사용
  - 그러나 콜백 패턴은 콜백 헬을 발생시켜 가독성을 나쁘게 하고, 에러의 예외 처리가 곤란하며, 여러 개의 비동기 처리를 한 번에 처리하는 데도 한계가 있다.

✅ 타이머 함수인 setTimeout, setInterval, HTTP 요청, 이벤트 핸들러는 비동기 처리 방식으로 동작

42.2 이벤트 루프와 태스크 큐

- 자바스크립트는 싱글 스레드 방식으로 동작하기 때문에 한 번에 하나의 태스크만 처리할 수 있지만, 브라우저가 동작하는 것을 보면 태스크가 동시에 처리되는 것처럼 느껴진다.
  - ex) HTTP 요청을 통해 서버로부터 데이터를 가져오면서 렌더링이 동시에 이루어짐
- **이벤트 루프** : 자바스크립트의 동시성을 지원

    - 이벤트 루프는 브라우저에 내장되어 있는 기능 중 하나
- 브라우저 환경을 그림으로 표현하면 아래와 같다.

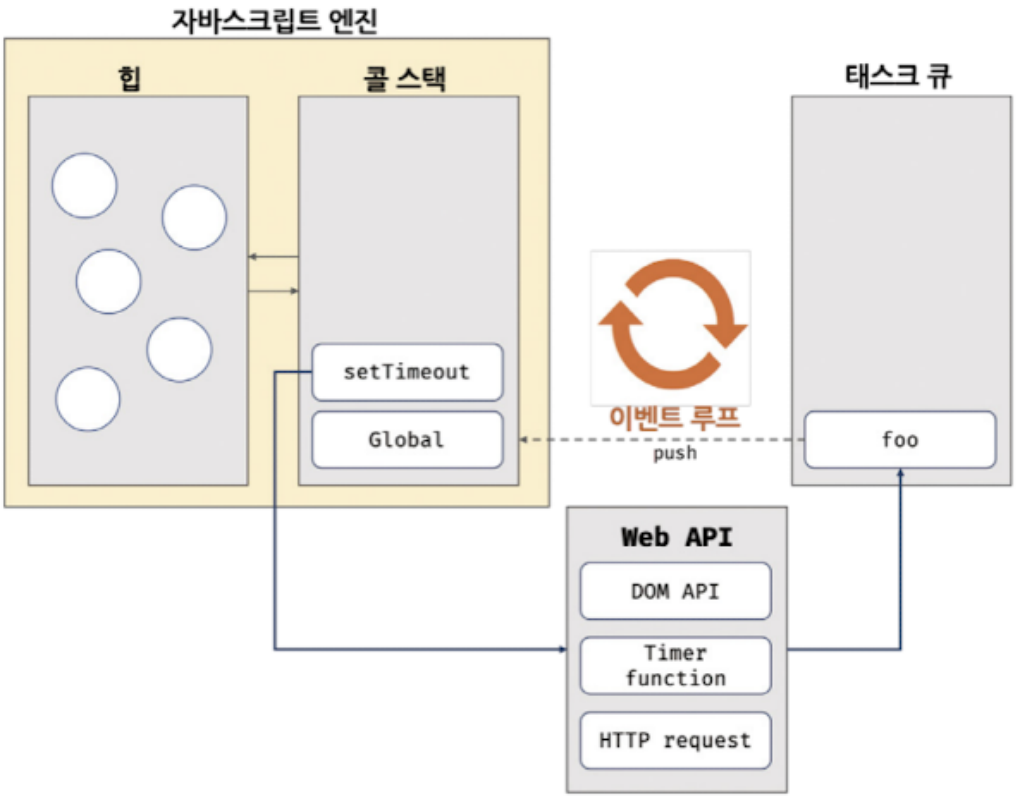

그림 42-4 이벤트 루프와 브라우저 환경

- 구글의 V8 자바스크립트 엔진을 비롯한 대부분 자바스크립트 엔진은 크게 2개의 영역으로 구분

> 💡 **콜 스택**
> 소스코드(전역 코드나 함수 코드 등) 평과 과정에서 생성된 실행 컨텍스트가 추가되고 제거되는 스택 자료구조인 실행 컨텍스트 스택
> 함수를 호출하면 함수 실행 컨텍스트가 순차적으로 콜 스택에 푸시되어 실행
> 자바스크립트 엔진은 단 하나의 콜 스택을 사용하기 때문에 최상위 실행 컨텍스트(실행 중인 실행 컨텍스트)가 종료되어 콜 스택에서 제거되기 전가지는 다른 어떤 테스크도 실행 ❌

💡 **힙**
객체가 저장되는 메모리 공간
콜 스택의 요소인 실행 컨텍스트는 힙에 저장된 객체를 참조
메모리에 값을 저장하려면 먼저 값을 저장할 메모리 공간의 크기를 결정해야 한다.
객체는 원시 값과는 달리 크기가 정해져 있지 않으므로 할당해야 할 메모리 공간의 크기를 런타임에 결정(동적 할당)해야 한다.
따라서 객체가 저장되는 메모리 공간인 힙은 구조화되어 있지 않다.

- 비동기 처리에서 소스코드의 평가와 실행을 제외한 모든 처리는 자바스크립트 엔진을 구동하는 환경인 브라우저 또는 Nods.js가 담당
- ex) 비동기 방식으로 동작하는 setTimeout의 콜백 함수의 평가와 실행은 자바스크립트 엔진이 담당하지만 호출 스케줄링을 위한 타이머 설정과 콜백 함수의 등록은 브라우저 또는 Node.js가 담당
- 이를 위해 브라우저 환경은 태스크 큐와 이벤트 루프를 제공

💡 **테스크 큐**
setTimeout이나 setInterval과 같은 비동기 함수의 콜백 함수 또는 이벤트 핸들러가 일시적으로 보관되는 영역
테스크 큐와는 별도로 프로미스의 후속 처리 메서드의 콜백 함수가 일시적으로 보관되는 마이크로테스크 큐도 존재

💡 **이벤트 루프**
콜 스택에 현재 실행 중인 실행 컨텍스트가 있는지, 그리고 테스크 큐에 대기 중인 함수(콜백 함수, 이벤트 핸들러 등)가 있는지 반복 확인
콜 스택이 비어 있고 테스크 큐에 대기 중인 함수가 있다면 이벤트 루프는 순차적으로 테스크 큐에 대기 중인 함수를 콜 스택으로 이동